C.C.# : 83118 ORG.# : RK-WP

AUTHOR: Roger Knights DATE : Aug. 23, 1983

SUBJECT: Turn-of-the-Century Problem

Received
DEC 0 8 1999
Director's Office
Group 2700

Here is a scenario to be avoided:

The 2000 phenomenon

In the last weeks of the year 1999 and in early 2000, business in the computerized economies of the world all but collapsed. For bills rendered in 1999 but due in 2000, overdue notices were issued already in 1999, charging the debtor interest for some 99 years. After January 1, 2000, many systems failed to issue overdue notices for amounts due in 1999 but not yet paid. The data in many reports were printed in the incorrect sequence: data pertaining to the year 2000 preceded data for the year 1999 instead of following it. Many computer runs terminated abnormally during this time because of overflow and similar errors. For software maintenance personnel this was a very difficult and trying time, during which the time pressure under which they worked was unusually severe.

The problems could all be attributed to the widespread use of a two-digit field for the year in computerized data files. In the late 1970's, the 1980's and the early 1990's, programmers had made incorrect assumptions regarding the operational lifetimes of the programs and the data files they designed.

It would seem desirable to add a clause that would shift the effective turn of the century far enough forward that a comparison or arithmetic operation between dates in different centuries could never be attempted. Let's say that a programmer codes "END CENTURY AT 09" in the entry for all two-digit year-fields in his program. 09 is ten years after 99. Therefore, prior to all comparisons or arithmetic operations involving that field, it will be moved to a hidden work area and ten will be subtracted from it, & 100 will be added if negative-result; then the operation will be executed. If the hidden field is the target of a math operation, it will have ten added back to it (100's truncated) after the operation, and then it will be moved back to the original source field. In effect, years are thus left-rotated by ten, temporarily.

As long as the years represented in a user's date fields are less than 100 years apart, it will be possible, using this proposed clause, for all later years to be effectively higher than all earlier years, regardless of the century they fall in. Note that if it is desired to allow a comparison of dates to be accurate for the maximum range (99 years), it would be necessary to update the END CENTURY operand every year. This is undesirable, and could be autonated by allowing the operand to take the form shown here:

END CENTURY AT CURRENT-YEAR + 1

Where CURRENT-YEAR contains the leftmost 2 digits of DATE.

DEC 0 8 1999

Director's Office
Group 2700

During any voluminous sorting operation, the inefficiency of this left-rotating would be very apparent; in fact, it would be desirable to avoid this inefficiency wherever possible. This means that the user will have to determine the maximum number of years his earliest year field will antedate his CURRENT-YEAR, and the number of years (max) his last field will postdate it. Say it's five in both cases; he codes:

```
END CENTURY AT CURRENT-YEAR + 5 WHEN CURRENT-YEAR IS 95 THRU 04
```

The effect of this clause is to activate the left-rotation of years only in the years xx95 thru xx04. Thus 90% of the time the inefficiency spoken of will be avoided.

Let's look at a hypothetical case. The year is 2000. The fields are DATE-ORDERED and WILL-BE-AVAILABLE-ON. It is a safe assumption that there are no open orders more than five years old, and that there are no out-of-stock situations that will persist for more than five years. The worst case would have a DATE-ORDERED of 95 and a WILL-BE-AVAILABLE-ON of 05. When these are left-rotated by 6 (2005-1999) they become 89 and 99. If the year is 1995, left-rotation of years 90 & 00 by 01 will convert them to 89 & 99, so they will compare as desired. If the year is 2004, left rotation of years 99 and 09 by 10 results in 89 & 99. Since this is the worst case, all other cases are safe.

Probably most cases would have a date-range of less than ten years, so that the length of time that inefficiency will have to be endured will be limited. For situations where the range of dates is longer, it might be worth expanding the fields to four digits. (Hmmm--maybe a four-digit year field in DATE will be needed too?) But shops should have the option of avoiding this disruptive file conversion if they desire.

It would be desirable to leave END CENTURY AT statements in place in programs to simplify things when 2100 rolls around. It would also be desirable to start putting such statements into programs well in advance of 2000, to avert a crisis-fix situation down the road. However, it should be possible to avoid the inefficiency that will result from every reference to a date field causing a check of CURRENT-YEAR. This isn't a serious inefficiency (unlike the one referred to above), but it seems "dumb" & offensive. Users who want to should be able to code a compiler-directing statement to ignore END CENTURY clauses when compiling UNLESS DATE > "900000" (for example). Such users would be trusting themselves to recompile frequently enough to enable CURRENT-YEAR checking in advance of its being needed, or they would be intending to keep records of the date-field characteristics of their programs with the intention of recompiling when needed, or both.

I'm certain the ideas here can be vastly improved on. I just want to start the ball rolling in hopes that someone else will run with it.

DE. 0 8 1999
Director's Office
Group 2700

One concern I have is how to get non-elementary date fields to inherit year-rotation from their year fields, without having inappropriate group items inherit this characteristic too, as when the group item containing a year field is not a date field.

Another concern is how this will fit in with the assembly-language date routines that some shops use for efficiency.

The latter concern would be reduced if COBOL had intrinsic date-handling functions to take over these tasks. Users who shifted over to using these functions would be working with a known quantity, which could be interfaced properly with the END CENTURY clause by the Committee.

Date-handling functions were suggested in (83030) Suggestion Smorgasbord, p. 2, #9. The advantages of using functions for this task were not fully enumerated there. They include:

Greater program portability (The program is more self-sufficient.)

Greater programmer portability (Less shop-specific stuff to learn.)

Greater efficiency for most shops

Less opportunity for computer crime. (Monkeying with an assembly-language date routine, or invoking a special copy of such a routine, are hard-to-detect ways a criminal might increase interest payments into his account, etc. Since such routines aren't standardized, detection would be harder for auditors. If the routine is in the language, monkeying will be harder & detection easier, I'd think.)

More features (The CC can do a fuller job than most shops.)

These advantages would apply to State-name functions too, mostly.

Another concern is reruns: CURRENT-YEAR might have to be adjusted to reflect some time in the past. Testing would necessitate adjustment of CURRENT-YEAR into the future.